論　文

API 呼出しを用いた動的バースマーク

岡本　圭司[†a)]　玉田　春昭[†b)]　中村　匡秀[†]　門田　暁人[†]
松本　健一[†]

Dynamic Software Birthmarks Based on API Calls

Keiji OKAMOTO[†a)], Haruaki TAMADA[†b)], Masahide NAKAMURA[†],
Akito MONDEN[†], and Ken-ichi MATSUMOTO[†]

あらまし　本論文では，ソフトウェア盗用の発見・立証を効率的に支援するための動的バースマークを提案する．動的バースマーク $f(p, I)$ はソフトウェア p を入力 I で実行した場合の p の振舞いから得られる，p のもつユニークな特徴の集合である．ソフトウェア p, q が存在し，$f(p, I) = f(q, I)$ であれば，q は p の盗用である疑いが強い．本論文では，2 種類の動的バースマークを提案する．提案手法では，ソフトウェア実行時の API 呼出し情報を，個々のソフトウェアの特徴として利用し，API 呼出しの順序から実行系列バースマークを，個々の API の呼出し頻度から実行頻度バースマークをそれぞれ抽出する．この 2 種類のバースマークを 2 種類の実験によって評価した．実験 1 では，同じ用途の複数のアプリケーションに提案バースマークを適用した．その結果，オリジナルとその改変アプリケーションからは非常に類似したバースマークが得られること，全く独立に実装されたアプリケーションからは，全く異なるバースマークが得られることが分かった．実験 2 では，提案バースマークは，異なるコンパイラや最適化などに対して強い耐性をもつことが示された．

キーワード　ソフトウェア盗用，著作権保護，バースマーク，API 呼出し，動的解析

1. まえがき

今日，ソフトウェアの盗用が世界中で問題となっている．代表的な事例として，SCO グループと IBM の法廷闘争 [1]，GPL 違反 [2]，フリーウェアやシェアウェアの著作権侵害 [3] などが報告されている．BSA の報告 [4] によると，2003 年のソフトウェアのコピーや著作権侵害による被害額は 2 億 8000 万ドルと推定されている．ソフトウェアの盗用は，ソフトウェア産業に重大な被害をもたらしている．

しかし，ソフトウェアを盗用から守ることは容易ではない．今日では無数のソフトウェアが開発，配布されているため，その中から盗用の疑いのあるソフトウェアを発見することは難しい．更に，盗用者がユーザインタフェースを変更したり，難読化ツール [5], [6] を用いた場合，ソフトウェアの盗用を発見及び立証することは極めて困難となる．

ソフトウェア盗用の発見・立証をより効率的に支援するために，ソフトウェアバースマーク [7]〜[9] と呼ばれる技術が提案されている．バースマーク（birthmark，あざ）とは直感的に，個々のソフトウェアが元来もっているユニークな特徴の集合を指す．プログラム p の（静的な）バースマーク $f(p)$ は，プログラム p の実行に不可欠な部分に，ある方法 f を適用することで抽出される．例えば，Java プログラムの場合では，クラスのフィールド変数の型や，継承関係，使用しているクラス等はプログラムごとにユニークな性質であり，バースマークとして利用できる [9]．ソフトウェア p と q に対し，$f(p) = f(q)$ が成立する場合，q は p のコピー（若しくはその逆）である疑いが強い．しかし，静的バースマークは，いくつかの攻撃に弱いことが示されている [8], [10]．

この問題に対して，ソフトウェア実行時の情報を用いる動的バースマークが提案されている [8], [11]．動的バースマーク $f(p, I)$ は，ソフトウェア p を p に対する入力 I をもとに実行したときの，p の振舞いから抽出される．Myles らは，動的バースマーク WPP

† 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–0101 Japan
a) E-mail: keiji@okamoto.ws
b) E-mail: harua-t@is.naist.jp

（Whole Program Path）を提案し，ごく小さな Java プログラムを用いて評価した [8]．WPP はプログラムの実行パスを特有の情報として抽出する．しかし，Myles ら自身が述べているように，WPP はループの平たん化やインライン展開といったプログラムの最適化技術に弱い．このことは，今日の Windows 環境のように，様々なコンパイラと多様な最適化オプションを有する開発環境において問題となる．同じソースコードを利用しソフトウェアを作成した場合でも，コンパイラとそのオプションを変えることで WPP は容易に改ざんされ得るのである．したがって，GPL ソフトウェアが盗用された場合など，盗用者がオリジナルソフトウェアのソースコードを入手した場合では，WPP を用いての盗用の発見・立証は難しい．また，福島らも Java プログラムに対する実行パス情報を使用した動的バースマークを提案しているが [11]，WPP と同様に，ループの平たん化など最適化技術に弱いと考えられる．更に，ネイティブコードで構成されるソフトウェアでは，その実行パス情報を収集することは非常に困難である．したがって，WPP 及び福島らの手法は，Windows 環境や Linux 環境などで動作するネイティブコードのソフトウェアでは利用することが難しいと考えられる．

この問題に対処するため，本論文では，特にネイティブコードを対象とした新たな 2 種類の動的バースマークを提案する．一般的に，中規模以上のソフトウェアは OS が提供する API（Application Program Interface）を用いて作成される．これは，OS が提供する機能を積極的に利用することで，容易にソフトウェアを作成できるからである．また，システム保護のため，いくつかの機能（ファイルの入出力など）は，特定の API を用いてしかアクセスすることができない．このような API は，等価な別の命令に置き換えることはできない．また，API の呼出しはコンパイラが最適化することもない．このことに着目し，提案手法ではプログラム p を入力 I で実行したときの API 呼出しの履歴を取得し，API の呼出し順序を実行系列バースマーク，各 API の呼出し頻度を実行頻度バースマークとして抽出する．本論文では，Windows 環境を対象に，実際にソフトウェアの実行時の API 呼出しを観測し，その情報から 2 種類の提案バースマークを抽出する方法を述べる．

また，本論文では，2 種類の実験を通じて提案バースマークを評価する．第 1 の実験では，五つの実用アプリケーションを使用し，提案手法の保存性と弁別性の評価を行った．ここで，保存性はコピーまたは改変により作成されたソフトウェアを同一と識別できる特性である．弁別性は独自に実装された異なるソフトウェアを異なるものと識別できる特性である．実験の結果，独立に開発されたソフトウェア同士では，提案バースマークによって低い類似度が示された．一方，オリジナルとその派生ソフトウェア同士では，提案バースマークによって非常に高い類似度が示された．提案手法はバースマークが満たすべき保存性と弁別性をもつことを確認した．第 2 の実験では，同一のソースコードを用いて，異なるコンパイラやコンパイルオプションで実行ファイルを作成し，コンパイラの最適化に対する提案バースマークの耐性を評価した．その結果，同一のソースコードから異なるコンパイラやコンパイルオプションで生成されたソフトウェアは，提案バースマークによって高い類似度が示された．異なるコンパイラやコンパイルオプションによる，提案手法への影響は軽微であることを確認した．

更に，提案手法の攻撃に対する耐性についての考察から，提案バースマークは機械的変換による攻撃に対する良好な耐性をもつことを示した．また，手作業による攻撃に対するある程度の耐性をもつことを示した．

2. 準　　備

2.1 コピー関係

議論を明確にするため，まず最初にソフトウェアのコピー関係を定義する．

[定義 1]（コピー関係） *Prog* を与えられたプログラムの集合とする．そして，$\equiv_{cp}$ を *Prog* 上の以下のような同値関係とする．$p, q \in Prog$ に対し，q が p のコピーであるならば，そのときに限り $p \equiv_{cp} q$ が成り立つ．このとき，$\equiv_{cp}$ をコピー関係と呼ぶ．

q が p のコピーであるかどうかの基準は絶対的なものはなく，状況に依存する．例えば，以下の基準は一般的なプログラムとしてはコピーであるといえる．

(a)　q は p の完全な複製である．

(b)　q は p のソースコード中に現れるすべてのシンボル名を変更したものである．

(c)　q は p のソースコードからコメント行をすべて削除したものである．

ここでは，混乱を避けるため，$\equiv_{cp}$ はユーザにより与えられるものとする．また，$\equiv_{cp}$ は同値関係であるから，以下の命題を満たす．いずれもコピーの概念と

直感的に一致する．

［命題 1］ $p, q, r \in Prog$ ならば以下の性質を満たす．

（反射律）$p \equiv_{cp} p$

（対称律）$p \equiv_{cp} q \Rightarrow q \equiv_{cp} p$

（推移律）$(p \equiv_{cp} q) \wedge (q \equiv_{cp} r) \Rightarrow p \equiv_{cp} r$

次に，q が p のコピーである場合の p と q の外面的な振舞いは同じであるから，以下が成り立つ．

［命題 2］ $Spec(p)$ を p の（外部）仕様とする．このとき，$p \equiv_{cp} q \Rightarrow Spec(p) = Spec(q)$ を満たす．

この命題の逆は必ずしも成り立たない．なぜなら，全く独立に同じ仕様のプログラムが実装されることがあり得るためである．

2.2 ソフトウェアバースマーク

与えられた $Prog$ と $\equiv_{cp}$ に対し，動的バースマークの概念を定義する．この定義は，Myles らによって与えられた [8]．

［定義 2］（動的バースマーク） p，q を与えられたプログラム，I を与えられた入力とし，$\equiv_{cp}$ を与えられたコピー関係とする．$f(p, I)$ を p とその入力 I からある方法 f により抽出された特徴の集合とする．このとき，以下の条件を満たすならば，$f(p, I)$ を p の動的バースマークであるという．

［条件 1］ $f(p, I)$ はプログラム p と p に対する入力 I のみから得られる

［条件 2］ $p \equiv_{cp} q \Rightarrow f(p, I) = f(q, I)$

条件 1 は，バースマークはプログラムの付加的な情報ではなく，p の実行に必要な情報であることを示す．すなわち，バースマークは電子透かしのように付加的な情報を必要としないことを表す．条件 2 はコピーされたプログラムからは同じバースマークが得られることを示す．対偶から，もしバースマーク $f(p, I)$ と $f(q, I)$ が異なっていれば，$p \not\equiv_{cp} q$ を満たす．これにより，q は p のコピーではないことが保証される．

2.3 バースマークの満たすべき性質

バースマークは以下の性質を満たすことが望まれる．

［保存性］ p から任意の等価変換により得られた p' に対して，$f(p, I) = f(p', I)$ を満たす

［弁別性］ $Spec(p) = Spec(q)$ となる p と q に対し，それらが全く独立に実装された場合，$f(p, I) \neq f(q, I)$ となる

保存性はプログラム変換によりバースマークが変化しないことを表す．攻撃者は盗用の事実を隠すために，もとのプログラムを等価な別のプログラムに変換することによりバースマークを改ざんする可能性がある．このような変換を行うための手法には，難読化 [5], [6], [12] や最適化があり，バースマークはこれらの攻撃によっても変化しないことが望ましい．一方，弁別性は全く独立に実装されたプログラム p，q を正しく区別できることを表す．一般的にプログラムがある程度の規模であれば，プログラムの詳細な部分まで一致することは非常にまれである．しかしながら，たとえ p と q が全く独立に実装されていても，プログラムの規模が非常に小さい場合，この性質を満たさない場合がある．一般的にすべてのプログラムに対して，保存性・弁別性を完全に満たすバースマークを設計することは難しい．したがって，実用上はユーザの判断により性質の強度を適宜決定する必要がある．

3. 提案手法

3.1 キーアイデア

まず，ソフトウェア内のどのような情報をバースマークとして利用すべきかを考える．例えば，ソフトウェアの「バイナリーコード中の XOR 命令の数」は，一見そのソフトウェアのユニークな性質と考えられる．しかし，ソフトウェアの盗用者は，XOR 命令をいくつかの AND 命令と OR 命令に書き換えることで，オリジナルソフトウェアの動作を保存しつつ，XOR 命令を消し去ることができる．したがって，この情報はバースマークとしては攻撃に弱く（fragile）実用的ではない．そこで，より改ざんされにくい情報として，我々はソフトウェアの API 呼出しに着目する．

OS 上で動作するソフトウェアは，API（Application Program Interface）を用いることで，OS の様々な機能を利用することができる．OS が API によって提供する機能としては，ファイル入出力機能，ウィンドウシステムなどのユーザインタフェース機能，セマフォやクリティカルセクションなどの同期機能などがある．

高度な OS は，システム保護の観点から，システムリソースへのアクセスを API を通じてしか許可しない．例えば，ファイル操作を行う場合では，ソフトウェアが直接ファイルシステムを操作することは許可せず，API による操作のみ許可する．このような API による命令は，他の等価な命令によって置き換えることができない．

提案手法では，2 種類のバースマークを提案する．第 1 の実行系列バースマークは，ソフトウェアの API 呼出し順序に基づくバースマークである．第 2 の実行

頻度バースマークは，API それぞれの呼出し回数に基づくバースマークである．

本論文で特に対象とするソフトウェアは，Windows 環境で動作する，API 呼出しをある程度利用している中規模以上のソフトウェアとする．API の利用が著しく少ないソフトウェアや，小規模なソフトウェアは対象外である．

3.2 提案バースマーク

3.2.1 実行系列バースマーク

p を与えられたプログラム，I を与えられた入力とし，I によって p を実行した場合の API 呼出し順序について考える．p を改変し，オリジナルの動作を維持しつつ API の呼出し順序を改変することは困難である．また，プログラム p と q が同様の API 群を利用している場合でも，API の呼出し順序まで同様になることはまれである．このことから，API の呼出し順序をバースマークとして利用する．

[定義 3]（実行系列バースマーク） p を与えられたプログラム，I を与えられた入力，W を与えられた API 関数の集合とする．$(w_1, w_2, \ldots, w_n)$ を，p を入力 I に基づいて実行させたときに呼び出される関数の（順序付きの）系列とする．このとき，w_i $(1 \leq i \leq n)$ のうち，W に属さないものを消去して得られる系列を，p の I に基づく実行系列バースマークと呼び，$EXESEQ(p, I)$ と表記する．

3.2.2 実行頻度バースマーク

攻撃者がソフトウェアの意味解析を行った上で API の呼出し順序を変更したとしても，ある単位時間及び単位機能の API 呼出し頻度は，ほぼ同様となる．このことから，API の呼出し頻度をバースマークとして利用する．

[定義 4]（実行頻度バースマーク） p を与えられたプログラム，I を与えられた p に対する入力，$(w_1, w_2, \ldots, w_n)$ を p の実行系列バースマーク（$EXESEQ(p, I)$）とする．$(w'_1, w'_2, \ldots, w'_m)$ を $EXESEQ(p, I)$ から関数名の重複をなくし，与えられた特定の順番に並べた系列とする．k_i $(1 \leq i \leq m)$ を w'_i の関数名，a_i を $EXESEQ(p, I)$ 中に現れる k_i の回数とする．このとき，$((k_1, a_1), (k_2, a_2), \ldots, (k_m, a_m))$ を p の I に基づく実行頻度バースマークと呼び，$EXEFREQ(p, I)$ と表記する．

3.3 バースマークの類似度

プログラム p，q とそれらに対する入力 I に対し，それぞれバースマーク $f(p, I) = (p_1, p_2, \ldots, p_n)$，$f(q, I) = (q_1, q_2, \ldots, q_n)$ が得られたとする．このとき，すべての i に対して $p_i = q_i$ である場合，$f(p, I)$ と $f(q, I)$ は同一のバースマークであるといい，$f(p, I) = f(q, I)$ と書く．この場合，一つでも $p_i \neq q_i$ の組があれば，ほかのすべてのペアが等しくても，$f(p, I) \neq f(q, I)$ となってしまう．よって，二つのバースマークが非常に似ているにもかかわらず，$p \not\equiv_{cp} q$ と結論づけてしまう．このように，全く同じかそうでないかでバースマークを比較すると，手法そのものが敏感になりすぎるため実用的ではない．

この問題に対処するため，我々は提案バースマークの類似度を定義する．実行系列バースマークでは，類似度として相互に含まれる部分系列の全体に対する比率を用いる．また，実行頻度バースマークでは，それぞれのバースマークをベクトルとし，ベクトル間の角度を類似度として用いる．

[定義 5]（実行系列バースマークの類似度） p，q を与えられたプログラム，I を与えられた p，q に対する入力とする．更に，p，q の実行系列バースマークをそれぞれ $EXESEQ(p, I) = \rho_p = (wp_1, wp_2, \ldots, wp_n)$，$EXESEQ(q, I) = \rho_q = (wq_1, wq_2, \ldots, wq_m)$ とする．ρ_p と ρ_q がともにある系列 $\rho = (r_1, r_2, \ldots, r_k)$ を含む場合，すなわち，$(r_1 = wp_i = wq_j), (r_2 = wp_{i+1} = wq_{j+1}), \ldots, (r_k = wp_{i+(k-1)} = wq_{j+(k-1)})$ なる i と j が存在する場合，ρ を $EXESEQ(p, I)$ と $EXESEQ(q, I)$ の一致列と呼ぶ．ρ_p と ρ_q の一致列のうち，最も長い一致列 ρ を最長一致列と呼ぶ．最長一致列が $\rho = (r_1, r_2, \ldots, r_k)$ のとき，k を最長一致列長と呼ぶ．このとき，$EXESEQ(p, I)$ と $EXESEQ(q, I)$ の類似度を以下の式で求める．

$$\frac{2k}{m+n}$$

この類似度は，$0 \leq k \leq n$，$0 \leq k \leq m$ であるため，0.0 から 1.0 までの値をとる．

[定義 6]（実行頻度バースマークの類似度） p，q を与えられたプログラム，I を与えられた p，q に対する入力とする．p，q の実行頻度バースマークをそれぞれ $EXEFREQ(p, I) = ((kp_1, ap_1) \ldots (kp_n, ap_n))$，$EXEFREQ(q, I) = ((kq_1, aq_1) \ldots (kq_n, aq_n))$ とする．$EXEFREQ(p, I)$，$EXEFREQ(q, I)$ から，$\vec{v_p} = [ap_1, ap_2, \ldots, ap_n]$，$\vec{v_q} = [aq_1, aq_2, \ldots, aq_n]$ なるベクトルを構成する．このとき，$EXEFREQ(p, I)$ と $EXEFREQ(q, I)$ の類似度を以下の式で求める．

$$\frac{ap_1aq_1 + ap_2aq_2 + \ldots + ap_naq_n}{\sqrt{ap_1^2 + ap_2^2 + \ldots + ap_n^2}\sqrt{aq_1^2 + aq_2^2 + \ldots + aq_n^2}}$$

これは，$\vec{v_p}$ と $\vec{v_q}$ のなす角度のコサインである．すべての i について p_i と q_i は正の整数であるため，類似度は 0.0 から 1.0 までの値をとる．

4. 実　　装

ソフトウェア実行時に API 呼出しを観測し，提案バースマークを抽出する手法を，MS-Windows 環境下で行う方法を述べる．実際のソフトウェアの盗用では，盗用ソフトウェアのソースコードを入手できることはまれであるため，ソースコードを用いることなくバースマークを抽出できることは重要である．本実装は，ソースコードを必要とせず，実行ファイルのみから提案バースマークを抽出する．

4.1 アウトライン

バースマークを抽出するためには，ソフトウェア実行時の API 呼出しを観測する必要がある．Windows では，Windows フックと呼ばれる機能で，ユーザインタフェース操作のためのメッセージ機構の監視，キーボード及びマウス入力の監視が可能であるが [13]，提案バースマーク抽出に必要な詳細情報の収集までは行うことができない．そこで，以下のような方針を採用する．

Step1： 実行中の対象ソフトウェアに対し，API 呼出し観測ルーチンを挿入する．

Step2： API 呼出しのための関数ポインタテーブルを変更する．

Step3： API 呼出しを記録する．

Step4： オリジナル API を呼び出す．

Step5： バースマークを抽出する．

4.2 実装の詳細

4.2.1 Step1：API 呼出し観測ルーチンの挿入

API 呼出しを観測するために，実行中の対象ソフトウェアに API 呼出し観測ルーチンを挿入する必要がある．ここでは，観測ルーチンを DLL（ダイナミックリンクライブラリ）として実装し，この DLL を実行中の対象ソフトウェアに強制的にロードするアプローチをとる（以降，挿入する DLL をパラサイト DLL と呼ぶ）．通常 Windows では，ソフトウェア自ら指定した DLL 若しくはシステムが指定した DLL しかロードすることができない．そこで，我々は Windows フックを用いて対象ソフトウェアにパラサイト DLL を強制的にロードさせる手法 [14] を用いる．図 1 に

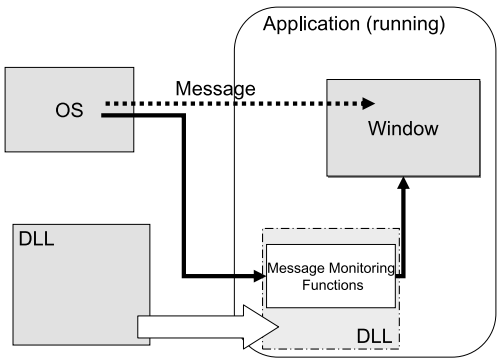

図 1　実行中のソフトウェアに DLL を挿入
Fig. 1　Loading a DLL in an application during runtime.

Windows フックの動作を示す．Windows のメッセージ機構は，通常システムからウィンドウにメッセージが直接送られるが（図 1：破線矢印），Windows フックを使用することによりメッセージを横取りすることができる（図 1：実線矢印）．Windows フックでは，メッセージの横取りを実現するために，指定したメッセージ監視関数を含む DLL を実行中のソフトウェアにロードさせる機能をもつ．本実装では，Windows フックの本来の目的であるメッセージの監視機能は使用せず，パラサイト DLL を強制的にロードさせる目的にのみ用いる．

4.2.2 Step2：API 関数ポインタテーブルの変更

実行中のソフトウェアが API を呼び出す際の動作は以下のとおりである．実行中のソフトウェアは，インポートセクションと呼ばれる領域をもつ．これは，API が実装されている DLL への関数ポインタを含むポインタテーブルである．図 2 の破線矢印で，通常の API 呼出しを示す．ソフトウェアが API を呼び出すと，呼び出した API に対応する関数ポインタをインポートセクションから得て，API が実行される．したがって，インポートセクションを変更すれば，ソフトウェアが API を呼び出したときに実行される関数を変更することができる [14]．DLL はロードされた時点で，その初期化コードが呼び出される．本実装では，パラサイト DLL の初期化コードでインポートセクションを変更する．具体的には，観測したい API に対応するインポートセクションのエントリを，パラサイト DLL に含まれる API 観測ルーチンへのポインタに書き換える．

4.2.3 Step3：API 呼出しの記録

Step2 でインポートセクションを書き換えたことに

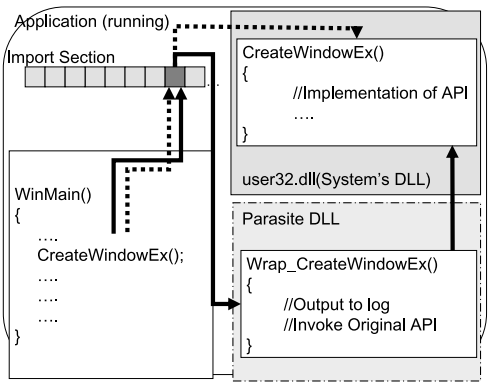

図 2　API 呼出しの観測
Fig. 2　Monitoring API calls.

より，実行中のソフトウェアが API を呼び出すと，本来呼び出される API 関数の代わりにパラサイト DLL の API 観測ルーチン（ラッパー関数）が呼び出される（図 2：実線矢印）．ラッパー関数では，以下の情報をログファイルに書き出す．

- 呼び出した API の種類（名前）
- API を呼び出した時間
- API を呼び出したスレッド ID

時間とスレッド ID は実行系列バースマークを抽出する際に用いられる．

4.2.4　Step4：オリジナル API の呼出し

対象ソフトウェアにパラサイト DLL を挿入しても，ソフトウェアのオリジナルの動作を維持しなければならない．そこで，パラサイト DLL のラッパー関数は，Step3 のあとにオリジナルの API 関数を呼び出し，その返り値をラッパー関数の返り値とする．図 2 の実線矢印で，対象ソフトウェアにパラサイト DLL が挿入された状態で API を呼び出した際の動作を示す．これにより，パラサイト DLL を挿入しても，ソフトウェアのオリジナルの動作は維持される．

4.2.5　Step5：バースマークの抽出

パラサイト DLL を挿入したソフトウェアを動作させることで，API 呼出しを観測し，情報を収集することができる．盗用が疑われる機能があれば，実際にユーザがその機能を操作，実行することで，該当個所のバースマークを得ることができる．得られたデータを解析し，API 呼出し記録をスレッド ID ごとにまとめ，時間順にソートすることで実行系列バースマークを得る．また，API 呼出しの出現回数から実行頻度バースマークを得る．

一般的に，マルチスレッドプログラムの実行系列をスレッドを区別せずに抽出した場合，複数のスレッドがどの順序で実行されるかは定まらないため，得られる実行系列も一意に定まらない．しかし，提案手法はスレッド ID により API を呼び出したスレッドを識別することができ，その呼出し系列をスレッド ID ごとに分類している．そのため，マルチスレッドに起因する実行形列の乱れは発生しない．

4.3　バースマーク抽出ツール – K2 Birthmark Tool Kit（K2BTK）

以上の実装アプローチに基づき，我々はバースマーク抽出ツール K2 Birthmark Tool Kit（K2BTK）[15] を実装した．K2BTK は，任意の Windows アプリケーションから，そのソースコードを用いることなく，実行系列バースマーク，実行頻度バースマークを抽出できる．また，実行系列バースマーク同士，実行頻度バースマーク同士の類似度を求めることができる．実装言語は，C++及び Delphi で約 11,000 行である．

MS-Windows はその基本 API として，Microsoft Platform SDK [16] に含まれる基本ヘッダファイル winbase.h で宣言された 613 種類を備えている．このうち，K2BTK では，以下の 14 種類の API を除いた 599 種類の API を観測可能である：EnterCriticalSection()，LeaveCriticalSection()，TLSGetValue()，RestoreLastError()，LocalLock()，LocalUnlock()，HeapAlloc()，HeapFree()，WriteProfileStringA()，LoadLibraryA()，LoadLibraryW()，LoadLibraryExA()，LoadLibraryExW()，GetProcAddress()．これら 14 種類の API は，ほとんどのアプリケーションにおいて呼出し回数が膨大になりすぎる API や，K2BTK の動作そのものに必要な API であり，現在の実装から意図的に除外されている．図 3 に K2BTK によって得られた API 呼出し系列の例を示す．

5.　評　　価

提案手法の有効性を評価するため，2 種類の実験を行った．実験の目的は，提案バースマークの保存性と弁別性を，同じ用途の実用ソフトウェア群を対象に評価すること，及び，コンパイラの最適化に対する提案バースマークの耐性を評価することである．

実験環境としては，ハードウェアとして DELL LATITUDE D600（Intel Mobile PentiumM 1.6 GHz, 1 GByte RAM）を使用した．OS は Microsoft Win-

dows XP Professional SP1 である．

5.1 実験 1：バースマークの保存性，弁別性の評価

本実験では，提案バースマークの保存性と弁別性の評価を行う．実験のために，同用途のソフトウェアを五つ用意し，提案バースマークによってそれらの類似度を評価した．

実験に使用したソフトウェア及びその作者を表 1 に示す．これらはすべて MP3 オーディオファイルのメタ情報タグを編集するソフトウェアである．Super Tage Editor 改（STE 改または STE-Ext）[17] は，Super Tag Editor 2.00b7（STE）[18] のソースコードを利用して作成された改変バージョンであり，作者がオリジナルの STE を合法的に引き継いで作成された．つゆたぐ（Tuyutag）[19]，Teatime [20]，Mp3tag [21] はそれぞれ STE とは全く独立に作成されたソフトウェアである．

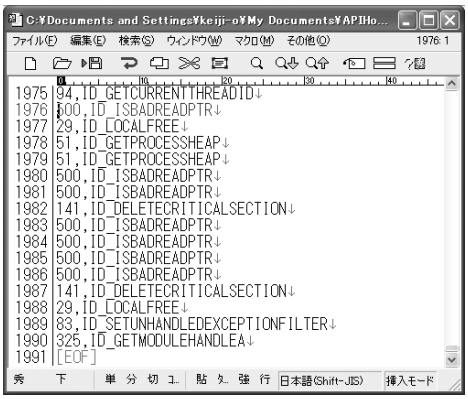

図 3　導出された API 系列の例
Fig. 3　An example of API sequence derived.

表 1　実験 1 で使用したソフトウェア
Table 1　Applications used in Experiment 1.

Applications	Authors
Super Tag Editor 2.00b7 (STE)	MERCURY
Super Tag Editor Extended Rev.32 (STE-Ext)	haseta
Tuyutag 2.02	P's Soft
Teatime 2.525	Toru Kuroda
Mp3tag 2.2.6.0	Florian Heidenreich

まず，参考実験として，五つの実行ファイルそのものの比較を行った．この比較のため，バイナリー差分作成ツール WDiff [22] を用いてバイナリーデータの差分を抽出し（WDiff の厳密差分モード 2 を使用），ファイルサイズに占める差分への割合を求めた．表 2 に各ソフトウェアのペアを比較した結果を示す．表中で，ファイルサイズ（File size）はそれぞれのソフトウェアの実行ファイルの大きさ（単位：バイト），差分サイズ（Diff size）は各ソフトウェアの実行ファイルを更新するために必要な差分データの大きさ（単位：バイト）である．また，割合（Percent）は比較する二つの実行ファイルの大きさのうち，大きな方に占める差分サイズの割合である．この結果から，実行ファイルをデータとして比較した場合には，それぞれのソフトウェアは全く類似性を示さないことが分かる．

次に，提案バースマークによる比較を行う．実験では，各ソフトウェアを起動したのち即終了し，起動から終了までに呼び出される API の呼出し履歴から，2 種類の提案バースマークを抽出する．そして，各ソフトウェアのバースマークをそれぞれ比較した．API 呼出し履歴の記録，バースマークの抽出及びバースマークの比較には K2BTK を用いた．K2BTK の性能の目安として，例えば STE と STE 改のバースマークの比較に必要な時間は，実行系列バースマークの比較に約 13 秒，実行頻度バースマークの比較に約 0.05 秒をそれぞれ要した．

表 3 に実行系列バースマークの結果を示す．この結果から，実行系列バースマークの類似度は，STE と STE 改との間で非常に高く，STE と STE 改以外との間ではいずれも低いことが分かる．

次に，図 4 に各ソフトウェアの実行頻度バースマークを示す．図中で，x 軸は API の種類，y 軸は各 API の呼出し回数の対数，z 軸はソフトウェアの種類である．図 4 から分かるように，STE と STE 改の実行頻

表 2　実行ファイルの差分サイズによる類似性評価
Table 2　Similarity differences of object code.

	STE-Ext		Tuyutag		TeaTime		Mp3tag		File size
	Diff size	Percent	Diff size	Percent	Diff size	Percent	Diff size	Percent	
STE	706,189	73.1%	903,345	94.4%	954,839	94.9%	1,626,527	89.8%	565,248
STE-Ext	—	—	902,370	93.3%	953,882	94.8%	1,612,652	89.1%	966,656
Tuyutag	—	—	—	—	771,803	76.7%	1,674,465	92.5%	956,928
TeaTime	—	—	—	—	—	—	1,674,023	92.5%	1,006,080
Mp3tag	—	—	—	—	—	—	—	—	1,810,432

表 3　実験 1 結果：実行系列バースマーク（EXESEQ）による類似度評価
Table 3　Result of Experiment 1: Similarity evaluation with EXESEQ.

	STE	**STE Ext**	**Tuyutag**	**TeaTime**	**Mp3tag**
STE	1.0000	0.9566	0.0107	0.0096	0.0019
STE-Ext	—	1.0000	0.0106	0.0095	0.0019
Tuyutag	—	—	1.0000	0.0115	0.0011
TeaTime	—	—	—	1.0000	0.0008
Mp3tag	—	—	—	—	1.0000

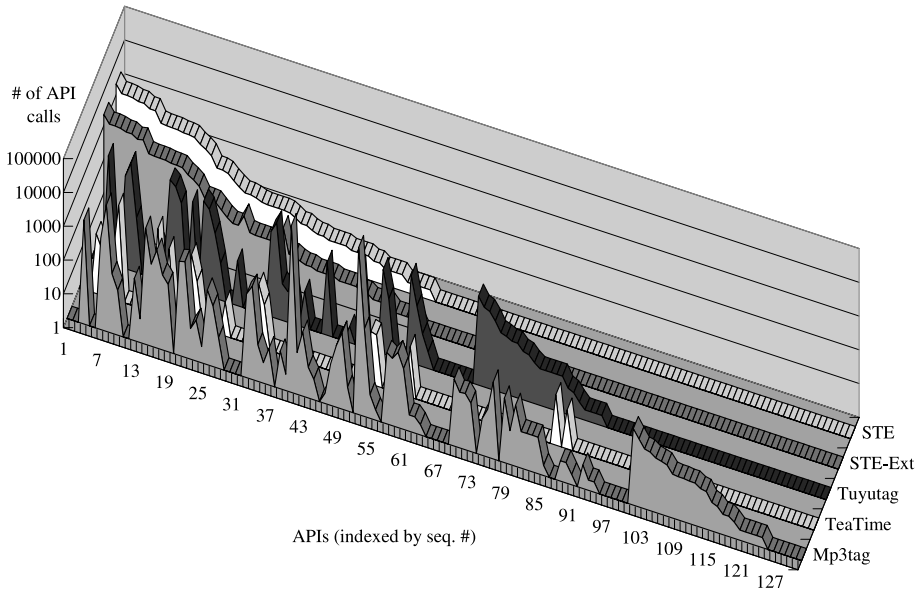

図 4　実行頻度バースマークのグラフ
Fig. 4　Frequency distribution of EXEFREQ.

表 4　実験 1 結果：実行頻度バースマーク（EXEFREQ）による類似度評価
Table 4　Result of Experiment 1: Similarity evaluation with EXEFREQ.

	STE	**STE Ext**	**Tuyutag**	**TeaTime**	**Mp3tag**
STE	1.0000	0.9997	0.3855	0.2891	0.0182
STE-Ext	—	1.0000	0.3960	0.3074	0.0191
Tuyutag	—	—	1.0000	0.7105	0.1157
TeaTime	—	—	—	1.0000	0.0837
Mp3tag	—	—	—	—	1.0000

度バースマークは非常に類似している．各ソフトウェア間の実行頻度バースマークによる類似度を表 4 に示す．この結果から，実行頻度バースマークの類似度は STE と STE 改との間で非常に高く，完全一致を示す 1.0 に極めて近いが，それ以外のソフトウェア間ではいずれも低いことが分かる(注1)．

このように，提案する実行系列バースマークと実行頻度バースマークは，同用途のソフトウェア群について，オリジナルとその改変バージョンである STE と STE 改との間で高い類似度を示した．一方，全く独立に実装されたソフトウェア間についてはいずれも低い類似度を示した．

またこのほかにも，ワードプロセッサである MS-Word2000 と一太郎 13，一太郎 2004 を相互に比較した．一太郎 13 と一太郎 2004 を比較した結果，EXESEQ，EXEFREQ の類似度はそれぞれ，0.855，0.999 となり，非常に高い類似性を示した．また，Word と

(注1)：表 4 においてつゆたぐと TeaTime に 0.7105 の類似度が確認できるが，実行頻度はコサイン類似度によって比較されるため，この値が高い類似度を表しているわけではない（図 4 参照）．実際 $\arccos(0.7105) \approx \pi/4$ であり，ベクトルのなす角としては完全一致の場合（$= 0$）と直行する場合（$= \pi/2$）の中間値となる．

一太郎の比較においては，EXESEQ の類似度は 0.001 以下，EXEFREQ については 0.133 以下となり，全く異なるバースマークをもつことが分かった．

我々が現在までに行ったケーススタディにおいて，提案バースマークは多くの実用プログラムに対して高い性能を示している．しかしながら，このことが必ずしもすべてのプログラムに対して有効と証明しているわけではない．今後，より多くのケーススタディを通して，提案バースマークの有効性の検証を続けていくことが重要である．

5.2 実験 2：コンパイラの最適化に対するバースマークの耐性

実験 2 では，プログラム最適化に対する，提案バースマークの耐性を評価する．具体的には，同一のソースコードから異なるコンパイラやコンパイルオプションを用いて実行ファイルを作成した場合，提案バースマークがどれだけ改変され得るかを調べる．

異なるコンパイラや，異なるコンパイルオプションを用いると，異なる実行ファイルが作成される．例えば，デバッグオプションを指定した場合，デバッグコードやシンボルテーブルなどが実行ファイルに付加される．また，リリースオプションを指定すると，コンパイラはソフトウェアが可能な限り高速に実行されるように，最適化を施す．この最適化技術は日々進歩しており，コンパイラのベンダーによってもノウハウが異なるため，異なるコンパイラが生成する実行ファイルは一般に異なる．Windows 環境は多数のコンパイラが存在するため，異なるコンパイラやコンパイルオプションを使って，バースマークを改ざんする攻撃が行われる可能性がある．特に，ソースコードが公開されている GPL ソフトウェアなどでは，このような攻撃を受ける可能性が高い．

実験では，実行ファイルを作成するために，Super Tag Editor 2.01 のソースコードを使用し，表 5 に示すコンパイラを用いてコンパイルを行った(注2)．以降，表中の略称を各コンパイラの呼称として使用する．

各コンパイラでは，デバッグビルド（Debug）とリリースビルド（Release）のためのコンパイルオプションを設定し，実行ファイルを作成した．デバッグビルドは，それぞれのコンパイラでデバッグビルドとしてデフォルトで用意されている設定とした．リリースビルドは，ソフトウェアの実行速度が可能な限り速くなるような設定とした．作成された実行ファイルのサイズは表 6 のようになった．これら六つの実行ファイルを提案バースマークを用いて比較する．なお，実験方法は実験 1 と同様である．

表 5　実験 2 で使用したコンパイラ一覧
Table 5　Compilers used in Experiment 2.

Compilers	Legend
IntelC++ 8.1.024_ev05(evaluation version)	IntelC
Microsoft VisualC++ 6.0 Professional Edition SP6	VC6
Microsoft VisualC++ 7.0 (VisualStudio.NET 2002)	VC7

表 6　実行ファイルのサイズ
Table 6　Size of executables.

Compilers and options	File size (bytes)
IntelC Debug	2,281,472
IntelC Release	913,408
VC6 Debug	1,699,960
VC6 Release	581,632
VC7 Debug	2,113,536
VC7 Release	577,536

実験 2 では，観測対象の API 集合 W から，以下の 4 種類を除外した：IsBadReadPtr()，IsBadWritePtr()，HeapValidate()，WaitForMultipleObjectsEx()．これらの API の呼出しは，OS の現状態（メモリ使用量や実行時間，実行中のプロセスなど）に非常に敏感であり，同一のソフトウェアを異なるタイミングで実行した場合でも，呼出し回数が大きく変動する．そこで，同一ソフトウェアの動的バースマークを同一に正規化するための予備実験を行い，これら 4 種類の API を除外することにした．

表 7 に実行系列バースマークの結果を示す．更に，表 8 に実行頻度バースマークの結果を示す．

表 7 から，すべての実行ファイルのペアに対し，実行系列バースマークは 0.92 以上の高い類似度を達成していることが分かる．特に，リリースビルド同士は，コンパイラが異なっても非常に類似している（0.97 以上）．また，実行系列バースマークにおける類似度の軽微な差異は，実行頻度バースマークで完全にマスクされている（表 8 参照）．このことから，API の実行順序を多少変更したとしても，実行頻度には影響しないことが分かる．表 8 の実行頻度バースマークの結果では，すべてのペアに対し，0.99 以上の高い類似度を達成している．

以上の結果から，異なるコンパイラやコンパイルオ

(注2)：実験 1 と同じバージョンである Super Tag Editor 2.00b7 のソースコードを用いるべきではあるが，2.00b7 のソースコードは提供されていないため，2.01 のソースコードを利用した．

表 7　実験 2 結果：実行系列バースマーク（EXESEQ）による類似度評価
Table 7　Result of Experiment 2: Similarity evaluation with EXESEQ.

	IntelC		VC6		VC7	
	Debug	**Release**	**Debug**	**Release**	**Debug**	**Release**
IntelC Debug	1.0000	0.9492	0.9243	0.9492	0.9988	0.9492
IntelC Release	—	1.0000	0.9392	0.9698	0.9492	0.9996
VC6 Debug	—	—	1.0000	0.9392	0.9243	0.9392
VC6 Release	—	—	—	1.0000	0.9492	0.9699
VC7 Debug	—	—	—	—	1.0000	0.9492
VC7 Release	—	—	—	—	—	1.0000

表 8　実験 2 結果：実行頻度バースマーク（EXEFREQ）による類似度評価
Table 8　Result of Experiment 2: Similarity evaluation with EXEFREQ.

	IntelC		VC6		VC7	
	Debug	**Release**	**Debug**	**Release**	**Debug**	**Release**
IntelC Debug	1.0000	0.9963	0.9966	0.9963	1.0000	0.9963
IntelC Release	—	1.0000	0.9912	1.0000	0.9963	1.0000
VC6 Debug	—	—	1.0000	0.9912	0.9966	0.9912
VC6 Release	—	—	—	1.0000	0.9963	1.0000
VC7 Debug	—	—	—	—	1.0000	0.9963
VC7 Release	—	—	—	—	—	1.0000

プションによる，提案バースマークへの影響は軽微であることが確認できた．

6. 考　　察

6.1 攻撃耐性

実際に盗用が行われた場合，盗用者は盗用の発覚をおそれるため，ソフトウェアに対し何らかの改変を加える可能性が高い．実験 2 で取り上げた，異なるコンパイラやコンパイルオプションを用いてソフトウェアを作成することも，改変による攻撃の一つである．こうしたバースマークへの攻撃は無数に考えられるが，ここでは機械的なプログラム変換と，手作業によるプログラム解析について，それぞれ定性的評価を行う．

6.1.1 難読化ツールなどの機械的変換に対する耐性

難読化 [12] とは，ソースコードやオブジェクトコードを理解しにくいものに等価変換することにより，ソフトウェアの解析や知的財産の盗用を防ぐための技術である．難読化は本来ソフトウェアの保護を行うものであるが，バースマークを改変する攻撃手法としても用いることができる．難読化ツール [5], [6] は，入力として与えられるソフトウェア（オブジェクトコードあるいはソースコード）を静的に解析して得られる情報を用いてソフトウェアを変換する．ただし，難読化対象のソフトウェアに外部ライブラリの呼出しが含まれる場合，外部ライブラリの動作を考慮した変換は行われない．

したがって，一般に難読化ツールは外部ライブラリの呼出しを別のものに置き換えたり，呼出しの順序を変更することはできない．そして，API 呼出しは外部ライブラリ呼出しである．そのため，提案手法は難読化ツールの影響を受けにくい．

また，難読化ツール以外の機械的変換についても，同様に外部ライブラリの呼出しを変更することはできない．よって，提案手法は機械的変換による影響を受けにくいと考えられる．

6.1.2 手作業による改変に対する耐性

攻撃者が手作業でソフトウェアの改変を行い，API 関数の機能を調べた上で，可能であれば等価なものに置き換えるなどして当該の API 呼出しを変更することができる．また，ソフトウェアの意味解析を行った上で，API 呼出しが含まれる部分の実行順序を変更したり，冗長な API 呼出しを追加するなどの攻撃が行われる可能性がある．

ここでは，手作業でオブジェクトコード及びソースコードを改変した場合に，提案バースマークによる識別結果に影響があるか否かを検討する．

まず，API 呼出しを削除する攻撃について述べる．別の API 若しくは命令群で置換え可能な API 呼出しは，その呼出しを削除し置き換えることが可能である．しかし，その API を使わなければ機能を実現できない API の呼出し，つまり置換え不可能な API の呼出しは，ソフトウェアの機能を維持しながら削除することは不可能である．したがって，観測対象の API を

置換え不可能なものに限定することにより，攻撃を難しくできる．

次に，API の呼出し順序を変更したり，冗長な API 呼出しを追加する攻撃について述べる．API の呼出し順序が変更されると，実行系列バースマークは異なるものになる．しかし，実行頻度バースマークは影響を受けない．冗長な API 呼出しが追加された場合では，実行系列バースマークと実行頻度バースマークはともに影響を受ける．この点は今後の検討課題である．解決策の一つとして，呼出しの増加による影響を最小化する新たな類似度算出方法を考えている．

6.2 動的バースマークと静的バースマークの比較

ここでは静的バースマークと動的バースマークを比較し，それぞれの利点と欠点について考察する．

まず，攻撃耐性については，動的バースマークの方が優れている．プログラムの仕様を変更せずに，実行時の挙動（動的情報）を変化させることは，プログラムの見た目（静的情報）を変化させるよりも困難なためである．

一方，バースマークの抽出容易性は，静的バースマークの方が優れている．動的バースマークの抽出は，プログラムを実際に動作させ，入力を与えるという作業が必要となるためである．したがって，動的バースマークは，短時間で多数のプログラムを比較する用途には向いていない．

最後に，バースマークの評価単位について述べる．静的バースマークは，ソフトウェア全体を単位として抽出・比較できるだけでなく，モジュール単位やクラス単位（Java の場合）で抽出・比較することが可能である [9]．対して，動的バースマークは，ソフトウェア全体に対して入力を与えてバースマークの抽出を行うため，評価，比較の単位は，ソフトウェア単位に限定される．

以上のように静的バースマークと動的バースマークにはそれぞれ利点と欠点がある．ただし，静的バースマークと動的バースマークは競合する技術ではなく，互いに補い合う関係にあるといえるため，併用することが望ましい．

7. むすび

本論文では，ソフトウェア盗用の発見及び立証を効率的に行うための技術として，ネイティブコードを対象とした 2 種類の動的バースマークを提案した．それぞれ，API の呼出し順序に注目した実行系列バースマークと，API それぞれの呼出し回数に注目した実行頻度バースマークである．そして，Windows 環境を対象に，実際にソフトウェアの実行時の API 呼出しを観測し，その情報から 2 種類の提案バースマークを抽出する方法を示した．

また，提案手法の有効性を評価するために，2 種類の実験を行った．実験 1 では，オリジナルのソフトウェアと派生ソフトウェア，同用途のソフトウェアであるが全く別個に作成された三つのソフトウェア，計五つを対象にし，実行ファイルの類似性及び提案バースマークによる類似度を評価した．その結果，提案バースマークは，これらの実用ソフトウェア群に対して十分な保存性・弁別性を有することを確認した．第 2 の実験では，同一のソースコードを用いて異なるコンパイラやコンパイルオプションによってソフトウェアを作成し，提案手法によるそれぞれの類似度について評価した．その結果，異なるコンパイラやコンパイルオプションによる提案バースマークへの影響は軽微であることを確認した．

更に，提案手法の攻撃に対する耐性について述べた．提案手法は，難読化などの機械的変換に対して耐性をもつことが期待される．また，手作業による攻撃に対しても，ある程度の耐性をもつことが期待される．

今後の課題として，置換え不可能な API のみを観測対象とすること，攻撃に強い新たな類似度算出手法を開発すること，などが挙げられる．更に，広範なソフトウェアに対して適用実験を行い，保存性・弁別性の統計的・客観的評価や，盗用と判断できる類似度の実用的なしきい値の決定などを行っていきたい．

謝辞 本研究の一部は，文部科学省科学研究費補助金（若手研究（B），課題番号：16700033）の補助を受けた．

文　献

[1] E. Raymond and R. Landley, “OSI position paper on the sco-vs.-ibm complaint,” 2004. http://www.opensource.org/sco-vs-ibm.html

[2] “Software encoder and decoder for iso mpeg-4 video,” 2002. http://www.sigmadesigns.com/products/RMP4_video_codec.htm

[3] T. Ueno, “PocketMascot に対する抗議ページ,” 2001. http://members.jcom.home.ne.jp/tomohiro-ueno/About_PocketMascot/About_PocketMascot.html

[4] Business Software Alliance, “Global software piracy study,” 2004. http://www.bsa.org/globalstudy/

[5] C. Collberg, “Sandmark: A tool for the study of software protection algorithms,” 2000. http://www.cs.arizona.edu/sandmark/
[6] “Codeshield Java byte code obfuscator,” 1999. http://www.codingart.com/codeshield.html
[7] D. Grover, ed., The protection of computer software — Its technology and applications, Second ed., The British Computer Society Monographs in Informatics, Cambridge University Press, 1992.
[8] G. Myles and C. Collberg, “Detecting software theft via whole program path birthmarks,” Proc. Information Security 7th International Conference, ISC 2004, vol.3225, pp.404–415, Springer-Verlag, Palo Alto, CA, USA, 2004.
[9] H. Tamada, M. Nakamura, A. Monden, and K. Matsumoto, “Design and evaluation of birthmarks for detecting theft of Java programs,” Proc. IASTED International Conference on Software Engineering (IASTED SE 2004), pp.569–575, Innsbruck, Austria, 2004.
[10] 福島和英，田端利宏，櫻井幸一，“Java バイトコードの静的解析によるプログラム指紋の抽出方法,” 情報処理学会研究報告，コンピュータセキュリティ研究会 2003-126, vol.126, pp.81–86, 2003.
[11] 福島和英，田端利宏，櫻井幸一，“動的解析を用いた Java クラスファイルのプログラム指紋抽出法,” 暗号と情報セキュリティシンポジウム 2004（SCIS 2004），vol.1, pp.1327–1332, 2004.
[12] C. Collberg and C. Thomborson, “Watermarking, tamper-proofing, and obfuscation—Tools for software protection,” IEEE Trans. Softw. Eng., vol.28, no.8, pp.735–746, 2002.
[13] Microsoft Corporation, “Microsoft Windows 32 ビット API リファレンス,” 1993.
[14] J. Richter, 長尾高弘，Advanced Windows 改訂第 4 版, Microsoft Press, ASCII, 2001.
[15] 岡本圭司，“K2BTK K2 Birthmark Tool kit,” 2005. http://se.naist.jp/K2BTk/
[16] Microsoft Corporation, “Microsoft Platform SDK.” http://www.microsoft.com/msdownload/platformsdk/sdkupdate/
[17] haseta: “Super Tag Editor 改,” 2004. http://hp.vector.co.jp/authors/VA012911/mp3DB/ste.html
[18] Mercury, “Super Tag Editor,” 2001. http://www5.wisnet.ne.jp/~mercury/supertag/index.html
[19] P’s soft: “MPEG ID3 TAG Editor つゆたぐ,” 2004. http://www.lares.dti.ne.jp/~mk3/Akmsoft/tuyutag.htm
[20] 黒田　徹，“Teatime,” 2002. http://hp.vector.co.jp/authors/VA011396/
[21] F. Heidenreich, “Mp3tag — The universal Tag Editor and more,” 2004. http://www.mp3tag.de/en/index.html
[22] 中川哲裕，“Wdiff,” 1998. http://www.vector.co.jp/soft/win95/util/se057654.html
[23] 岡本圭司，玉田春昭，中村匡秀，門田暁人，松本健一，“ソフトウェア実行時の API 呼び出し履歴に基づく動的バースマークの提案,” ソフトウェア工学の基礎 XI，日本ソフトウェア科学会 FOSE2004, pp.85–88, 2004.
[24] 岡本圭司，玉田春昭，中村匡秀，門田暁人，松本健一，“ソフトウェア実行時の API 呼び出し履歴に基づく動的バースマークの実験的評価,” 第 46 回プログラミング・シンポジウム報告集，pp.41–50, 2005.

（平成 17 年 10 月 6 日受付，18 年 2 月 28 日再受付）

岡本　圭司　（正員）

平 13 奈良工業高等専門学校・情報卒．平 15 奈良工業高等専門学校専攻科・電子情報卒．平 17 奈良先端科学技術大学院大学情報科学研究科博士前期課程了．ソフトウェアプロテクション，文書鑑定の研究に従事．

玉田　春昭　（正員）

平 11 京都産業大・工・情報通信卒．平 13 同大大学院博士前期課程了．同年住商エレクトロニクス（株）入社．平 18 奈良先端科学技術大学院大学情報科学研究科博士後期課程了．同年同大学産学官連携研究員．博士（工学）．ソフトウェアプロテクション，エンタープライズアプリケーションの研究に従事．IEEE 学生会員．

中村　匡秀　（正員）

平 6 阪大・基礎工・情報卒．平 8 同大大学院博士前期課程了．平 11 同大学院博士後期課程了．同年カナダ・オタワ大学ポスドクフェロー．平 12 阪大・サイバーメディアセンター助手．平 14 奈良先端大・情報科学・助手．博士（工学）．通信ソフトウェアの検証，サービス競合，ソフトウェアプロテクション等の研究に従事．IEEE 会員．

門田　暁人　（正員）

平 6 名大・工・電気卒．平 10 奈良先端大・情報・博士了．同年同大助手．平 16 同大助教授．平 15〜16 オークランド大学客員研究員．博士（工学）．ソフトウェア工学，ソフトウェアプロテクション等の研究に従事．IEEE，ACM，IPSJ，JSSST，JSiSE 各会員．

松本　健一　（正員）

昭 60 阪大・基礎工・情報卒．平元同大大学院博士課程中退．同年同大・基礎工・情報・助手．平 5 奈良先端科学技術大学院大学・情報科学・助教授．平成 13 年同大・情報科学・教授．工博．ソフトウェア品質保証，ユーザインタフェース，ソフトウェアプロセス等の研究に従事．情報処理学会，IEEE，ACM 各会員．